1966

Three characterizations of algorithm

Elmer Luis Schuman
The University of Montana

129

THREE CHARACTERIZATIONS OF ALGORITHM

By

Elmer Luis Schuman

B.A. Reed College 1963

Presented in partial fulfillment of the requirements

for the degree of

Master of Arts

UNIVERSITY OF MONTANA

1966

Approved by:

David R Anderburn
Chairman, Board of Examiners

Fred P Hornbake
Dean, Graduate School

AUG 15 1966
Date

ACKNOWLEDGEMENTS

I wish to express my appreciation to Professor David Arterburn for his interest, guidance, and assistance in the writing of this thesis. I also wish to thank Professors Merle Manis and William M. Myers for their assistance and critical reading of the paper.

E.L.S.

TABLE OF CONTENTS

INTRODUCTION

The generalization of concepts and of solutions to problems is a common activity of mathematicians. An instance of this is the development of a method for solving a given class of problems after the solution of a particular problem of the class has been found. A well known example of this is the algorithm for determining the greatest common divisor of two positive integers. The processes for determining the sum and product of two integers are other examples of algorithms or methods for solving a given class of problems.

The logical consequence of considering this type of generalization is the posing of the following problem.

> "Construct an algorithm for solving any mathematical problem."

The mathematician will react more violently to such a proposal than someone whose interests lie elsewhere. This is probably because he is aware of the inclusiveness of the phrase "any mathematical problem" and also because he refuses to believe that his job can be handed over to anyone who can simply follow a list of instructions. Accordingly, the mathematician provides the

following criticisms of the posing of this problem.

(1) What is the meaning of any mathematical problem?

(2) What, precisely, is an algorithm?

(3) Even if the first two criticisms are answered, there may be classes of problems for which no algorithm can be found because none exists.

This paper is concerned with the last two of these criticisms.

A series of investigations was undertaken beginning in the 1930's for characterizing algorithms. Various mathematical logicians proceeded from different conceptions of the problem and arrived at different definitions. This paper is an exposition of the methods of three of these investigators: S. C. Kleene, A. Church, and A. Turing. The main points of interest are:

(1) The methods which were developed to deal with such a general and, at that time, ill-defined concept.

(2) Each of the investigators answered the third criticism by exhibiting a class of problems for which no algorithm exists. This indicates that mathematical thinking must remain creative.

(3) The three characterizations developed by these investigators, although strikingly differing concepts, turn out to be demonstrably equivalent. This fact yields heuristic evidence for the validity of the

characterizations.

The reader is asked to note that the three mathematicians, whose ideas are used in this paper, were concerned in their work with questions much broader than those discussed here. This paper is not a summary of their work, but only an interpretation of their methods as they apply to the immediate problem of characterizing algorithms.

The problem of characterizing all algorithms is too large. In this paper, the problem is reduced to considering functions whose domain, and range, is the set of non-negative integers, M. An attempt will be made to characterize all such functions which are constructible in the following sense.

Definition: A function $\emptyset$ whose domain and range is M is said to be constructible if, for each k-tuple $n_1, \ldots, n_k$ $(k = 1, 2, \ldots)$ of non-negative integers, the non-negative integer $\emptyset(n_1, \ldots, n_k)$ can be determined in a finite number of steps. A characterization of constructible functions is essentially a characterization of arithmetical algorithms. As an example, addition may be considered as a function of two variables. If $\emptyset(a,b) = a + b$, $\emptyset$ is a constructible function since, given two non-negative integers, the integer which is their sum

can be determined in a finite number of steps. Not every function whose domain, and range is M is necessarily constructible. Let $A \subset M$ and let χ_A be the characteristic function of A:

$$\chi_A(n) = \begin{cases} 1 \text{ if } n \in A \\ 0 \text{ if } n \notin A \end{cases}$$

If A is infinite and there is no method of determining membership in A except by serially examining its members, $\chi_A(n)$ may not be determinable in a finite number of steps.

The reader is asked to note that, contrary to the usual usage, the statement "the domain, and range, of $\emptyset$ is M" means only that $\emptyset(M) \subset M$ and not necessarily that $\emptyset(M) = M$.

CHAPTER 1

GENERAL RECURSIVE FUNCTIONS

The definition of constructible function given above is not rigorous or precise. It does not define what is meant by determination in a finite number of steps. The notion of a function being constructible is purely intuitive and requires formalization if it is to be studied. The particular formalization that will be discussed in this chapter is based on the following idea studied by Kleene [3].

(1) One admits into the constructible category a finite number of definite types of functions. These initial functions are somewhat like the postulates for a geometry. They act as the link between a purely formal system and the intuitive notion which the system is to represent.

(2) Next, certain formal procedures are defined by which the initial functions may be used in combinations to produce additional functions. More precisely, the procedure is as follows.

Definition: $\emptyset$ is an initial function if it is defined as one of the following types:

I-1: $\phi(n) = n'$ (the successor function)

I-2: $\phi(n_1, \ldots, n_k) = m$ (constant functions)

I-3: $\phi(n_1, \ldots, n_k) = n_i$, $i \leq i \leq k$ (projection onto the i-th coordinate)

Two schemes are defined for forming additional functions.

S-1: $\phi(n_1, \ldots, n_k) = \psi(\chi_1(n_1, \ldots, n_k), \ldots \chi_\ell(n_1, \ldots, n_k))$

(composition of functions)

S-2: $\phi(0, n_2, \ldots, n_k) = \psi(n_2, \ldots, n_k)$ (= m, a constant if k = 1)

$\phi(m', n_2, \ldots, n_k) = \chi(m, \phi(m, n_2, \ldots, n_k), n_2, \ldots, n_k)$

(definition by induction)

Definition: A function is primitive recursive if it can be defined by zero or more applications of S-1 and S-2 to the initial functions.

Definition: Let $n \in M$ and let $P(n)$ be any property of n. Then $\varepsilon\, n[P(n)]$ denotes "the least n such that P(n)". This notation will be used only when there is at least one n with P(n).

Definition: A function is general recursive if it can

be expressed in the form $\phi(\varepsilon\, m[\Psi(n_1, \ldots, n_k, m) = 0])$ where ϕ, Ψ are primitive recursive and for each k-tuple of numbers in M, there is at least one m such that $\Psi(n_1, \ldots, n_k, m) = 0$.

It is clear that every primitive recursive function is general recursive, since, given any primitive recursive function $\Psi(n_1, \ldots, n_k)$, we can express the fact that $\Psi(n_1, \ldots, n_k) = p$ as $\phi(\varepsilon\, m[|\Psi(n_1, \ldots, n_k) - m| = 0]) = p$ by letting ϕ be the identity function, because then ϕ is primitive recursive and, as will be shown below, the function $\chi(n_1, n_2)$ defined by $\chi(n_1, n_2) = m$ if, and only if, $|n_1 - n_2| = m$, is primitive recursive.

It will now be shown that some of the common constructible functions of arithmetic are general recursive. Consider the function $\phi(b, a)$ defined by

$\phi(b, a) = m$ if, and only if, $a + b = m$.

ϕ can be shown to be primitive recursive and hence general recursive by using scheme S-2:

$$\begin{cases} \phi(0, a) = a \\ \phi(m', a) = [\phi(m, a)]' \end{cases}$$

If $\phi(b, a)$ is abbreviated to $a + b$, this may be written

as

$$\begin{cases} a + 0 = a \\ a + m' = (a+m)'. \end{cases}$$

Next, suppose $\Theta(b, a)$ is defined by

$\Theta(b, a) = m$ if, and only if, $a \cdot b = m$.

Using S-2, we can write

$$\begin{cases} \Theta(0, a) = 0 \\ \Theta(m', a) = \emptyset(a, \Theta(m, a)) \end{cases}$$

If $\Theta(b, a)$ is abbreviated to $a \cdot b$, this is

$$\begin{cases} a \cdot 0 = 0 \\ a \cdot b' = a \cdot b + a \end{cases}$$

Using abbreviations only, one may build upon previous definitions as follows (Kleene [5]).

$$a^b: \begin{cases} a^0 = 1 \\ a^{b'} = a^b \cdot a \end{cases}$$

$$a!: \begin{cases} 0! = 1 \\ a'! = a! \cdot a' \end{cases}$$

predecessor of a:
$$\begin{cases} pd(0) = 0 \\ pd(a') = a \end{cases}$$

a-b if $a \geq b$, 0 if $a < b$:
$$\begin{cases} a \dot{-} 0 = a \\ a \dot{-} b' = pd(a \dot{-} b) \end{cases}$$

$\min(a, b) = b \dot{-} (b \dot{-} a)$

$\min(a_1, \ldots, a_n) = \min(\ldots \min(\min(a_1, a_2), a_3) \ldots, a_n)$

$\max(a, b) = (a + b) \dot{-} \min(a, b)$

$|a - b| = |a \dot{-} b) + (b \dot{-} a)$

These examples make the following statement plausible.

T: Every constructible function is general recursive, and conversely.

The statement T, often called Church's Thesis, can by its very nature never be proven. However, under the assumption that T is true, it can be shown that there exist mathematical problems whose solutions are not in the algorithmic or constructible category. The procedure for accomplishing this (due to Post [6]) is as follows. Consider a subset A of M, the set of non-negative integers. The decision problem for A is whether or not there exists an algorithm (a method involving only a finite number of steps) for determining whether or not a given element of M is also an element of A.

Definition: A subset A of M is recursive if the characteristic function of A is general recursive.

Definition: The decision problem for A is recursively

solvable if A is recursive.

Definition: A subset A of M is recursively enumerable (r.e.) if there exists a general recursive function $\phi(n)$ whose range is exactly A. The sequence $\phi(1)$, $\phi(2)$, ... is called a recursive enumeration of A.

It will now be shown that there exists a subset of M whose decision problem is not recursively solvable. (Note that this implies the existence of a function which is not general recursive, namely the characteristic function of this subset.)

Lemma: A set $A \subset M$ is recursive if, and only if, both A and its complement A' are r.e.

Proof: Suppose first that A is finite. Then the characteristic function of A, χ_A is constructible. In fact, let $A = \{n_1, \ldots, n_k\}$ and let $n \in M$. Then, to determine $\chi_A(n)$, at most k steps are necessary, each one consisting of comparing n with $n_1, \ldots, n_k$. Thus, by the assumption T, χ_A is general recursive and hence A is recursive. Also, one may show that A and A' are r.e. as follows. Let

$$\begin{cases} \phi(1) = n_1 \\ \phi(m') = \phi_1(m) \end{cases}$$

$$\begin{cases} \phi_1(1) = n_2 \\ \phi_1(m') = \phi_2(m) \end{cases}$$

$$\vdots$$

$$\begin{cases} \phi_{k-1}(1) = n_k \\ \phi_{k-1}(m') = n_k \end{cases}$$

Then

$$\phi(1) = n$$

$$\phi(2) = \phi_1(1) = n_2$$

$$\vdots$$

$$\phi(k) = \phi_1(k-1) = \ldots = \phi_{k-1}(1) = n_k$$

$$\vdots$$

$$\phi(k+p) = \phi_1(k+p-1) = \ldots = \phi_{k-1}(1+p) = n_k$$

Thus $\phi(1), \phi(2), \ldots$ is a recursive enumeration of A and hence A is r.e. Now let $m = \max(n_1, \ldots, n_k)$. Then if:

$B = \{n \in M | n < m, n \notin A\} = \{m_1, m_2, \ldots, m_p\}$ and

$\overline{B} = \{m + 1, m + 2, \ldots\}$, then $A' = B \cup \overline{B}$.

Define as before

$$\begin{cases} \theta(1) = m_1 \\ \theta(x') = \theta_1(x) \end{cases}$$

$$\left\{\begin{array}{l}\theta_{p-1}(1) = m_p \\ \theta_{p-1}(x') = n_k + x - 1.\end{array}\right.$$

Then

$$\theta(1) = m_1$$
$$\theta(2) = \theta_1(1) = m_2$$
$$\theta(p) = \theta_1(p-1) = \ldots = \theta_{p-1}(1) = m_p$$
$$\theta(p+q) = \theta_1(p+q-1) = \ldots = \theta_{p-1}(1+p) = m + p$$

Thus $\theta(1)$, $\theta(2)$, ... is a recursive enumeration of A' and hence A' is r.e. Similarly, the lemma is true if A' is finite. Finally, suppose that both A and A' are infinite. If A is recursive, each integer 0, 1, 2, ..., as it is generated by the successor function, can be checked for membership in A with its characteristic function. This is a constructible process for generating the elements of A and hence, under the assumption T (interpreted in the sense that if one can enumerate a set constructively then there is a general recursive function which enumerates the set) there exists a general

recursive function which generates A, and hence A is r.e. Conversely, if both A and A^0 are r.e., let

$n_1, n_2, \ldots$ be a recursive enumeration of A and

$m_1, m_2, \ldots$ be a recursive enumeration of A^0.

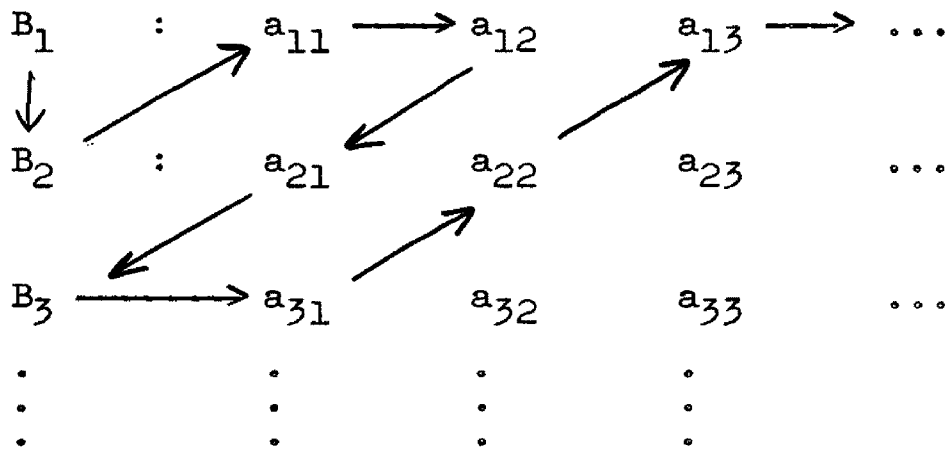
Now, given $n \in M$, check through the sequence $n_1, m_1, n_2, m_2, \ldots$. Since $n \in A$ or $n \in A^0$, after a finite number of steps n will be found to be in A or in A^0. This is a constructible method for determining membership in A, hence A has a general recursive characteristic function and A is recursive.

Theorem: There exists a r.e. subset of M whose decision problem is not recursively solvable.

Proof: By the lemma, this is equivalent to the existence of a r.e. subset A of M such that A^0 is not r.e. Post [6] has shown that the collection of r.e. subsets of M is enumerable and that each r.e. set A_i may be thought of as having a "base" B_i which determines its elements and that the collection of these "bases" can be constructively generated. In other words, if a_{ij} denotes the j-th element in a recursive enumeration of A_i, the infinite array

$$\begin{array}{cccc} a_{11} & a_{12} & a_{13} & \cdots \\ a_{21} & a_{22} & a_{23} & \cdots \\ a_{31} & a_{32} & a_{33} & \cdots \\ \vdots & \vdots & \vdots & \end{array}$$

contains all non-negative integers which appear in some r.e. subset of M. Let $B = \{n \in M \mid n = a_{nj} \text{ for some } j = 1, 2, \ldots\}$. Then B can be constructively enumerated as follows. The array is enumerated,



that is, as each B_i is generated, the process which enumerates the elements of A_i is begun. Now, whenever $a_{nj} = n$, n is placed in a set B. This is a constructible method for enumerating the elements of B and hence B is r.e. Finally, consider B', the complement of B. Given any r.e. subset A_m of M, $m \in A_m$ if, and only if, $m \notin B'$. Thus B' differs from each r.e. set.

Note that Post's result that the collection of r.e. sets is enumerable can be used directly to show the existence of a subset of M which is not r.e. by a simple cardinality argument. This particular theorem, however, has the following application.

The theorem, although it is a result about a very small part of the collection of all mathematical problems, can be applied to show the impossibility of algorithmic solutions to other classes of problems. This may be done by associating positive integers with the problems of a given class using a technique developed by Gödel. An example of this will be carried through in Chapter II. A more immediate application of this theorem is as follows. Consider the set D of all diophantine equations with integer coefficients which have solutions in integers. This set can be constructively generated as follows. There are enumerably many diophantine equations with integer coefficients,

$$d_1, d_2, d_3, \ldots$$

Since the domain of their variables is also an enumerable set, we have the array

$$
\begin{array}{lcllll}
d_1 & : & d_{11} & d_{12} & d_{13} & \cdots \\
d_2 & : & d_{21} & d_{22} & d_{23} & \cdots \\
d_3 & : & d_{31} & d_{32} & d_{33} & \cdots \\
\vdots & & \vdots & \vdots & \vdots &
\end{array}
$$

where the i-th row represents the result of all possible substitutions for the variables of d_i. If the array is listed in the serial order used in the proof of the theorem, and, whenever a true equation d_{ij} is generated, d_i is placed in D. The set A of subscripts of elements of D becomes a r.e. set of non-negative integers. In this manner, the problem of determining for an arbitrary diophantine equation with integer coefficients whether or not it has a solution in integers has been reduced to the decision problem for A.

CHAPTER II

λ-DEFINABILITY

In Chapter I, a possible formalization of the intuitive notion of constructible function was given in the form of general recursive function. That formalization consists of systematically defining certain types of functions. The concept of λ-definability, devised by Church [1] (and Kleene), is, on the other hand, an attempt to formalize the notion of constructible function by a particular formalization of the concept of a function. The resulting formal system is usually referred to as Church's calculus of λ-conversion.

The symbols used in this system consist of three types of brackets $\{ , \}$, (,), [,], the symbol λ , and an enumerably infinite set of symbols a, b, c, ... called variables.

Definition: A formula is any finite sequence of the symbols listed above.

Definition: A variable x in a formula is bound in that formula if it appears immediately after the symbol λ . Otherwise, the variable is free.

Definition: A well formed formula (w.f.f.) is defined inductively:

(1) A variable is a w.f.f.

(2) If F and X are w.f.f.s then $\{F\}(X)$ is a w.f.f.

(3) If M is a w.f.f. and x is a varialbe, which occurs free in M, then $\lambda x[M]$ is a w.f.f.

Certain abbreviations are convenient for writing w.f.f.s. Let the symbol $\longrightarrow$ be understood as $\alpha \longrightarrow A$ means "α is an abbreviation for A".

$F(X) \longrightarrow \{F\}(X)$

$F(X,Y) \longrightarrow \{\{F\}(X)\}(Y)$

$F(X,Y,Z) \longrightarrow \{\{\{F\}(X)\}(Y)\}(Z)$

$\vdots$

$\lambda x_1 \ldots x_n \cdot M \longrightarrow \lambda x_1[\lambda x_2[\ldots \lambda x_n[M]\ldots]]$

$S_N^x M| \longrightarrow$ the formula which results when the formula N is substituted for each occurrence of x in the formula M.

Three operations are defined on w.f.f.s:

I. To replace any part $\lambda x[M]$ of a formula by $[S_y^x M|]$ where y is a variable which does not occur

in M.

II. To replace any part $\{\lambda x[M]\}(N)$ of a formula by $S^x_N M|$ provided that the bound variables in M are distinct from both x and from the free variables in N.

III. To replace any part $S^x_N M|$ (not immediately following λ) of a formula by $\{\lambda x[M]\}(N)$, provided that the bound varialbles in M are distinct both from x and from the free variables in N.

Definition: If A and B are formulas, A is convertible into B (A conv B) if B is obtainable from A by a finite number of successive applications of I, II, and III.

The motivation for these definitions may be understood by considering the following intuitive interpretation of w.f.f.s. $\{F\}(X)$ may be thought of as "the function F of the argument X" and $\lambda x[M]$ as "that function which M is of x". Operation I can be interpreted as a change of variable. Operation II as the evaluation of the function that M is of x at the argument N; operation III is the reverse of operation II.

The following abbreviations and definition relates the calculus of λ-conversion to functions whose domain, and range, is the set of non-negative integers.

$0 \longrightarrow \quad ab\cdot a(b)$

$1 \longrightarrow \quad ab\cdot a(a(b))$

$2 \longrightarrow \quad ab\cdot a(a(a(b)))$

$\vdots$

Definition: A function $\emptyset(n_1, \ldots, n_k)$ defined for k-tuples of non-negative integers $n_1, \ldots, n_k$ is λ-definable if there is a formula F such that whenever $\emptyset(n_1, \ldots, n_k) = n_o$, $F(N_1, \ldots, N_k)$ conv N_o (and to no other formula representing a numeral) where N_i is the formula of which n_i is an abbreviation.

Parallel to the development in Chapter I, it will now be shown that some of the common constructible functions are λ-definable. These examples will provide some evidence in favor of associating constructible functions with λ-definable functions (the strongest evidence for this will be provided in Chapter IV, where general recursive and λ-definable are shown to be equivalent concepts).

Consider the formula $I \longrightarrow \lambda a[a]$:

$I(N) \longrightarrow \{I\}(N) \longrightarrow \{\lambda a[a]\}(N)$

conv $S_N^a a|$ which is N.

Thus if N is a formula which represents a non-negative

integer (henceforth N, N_i will always denote formulas which represent non-negative integers n, n_i),

$$I(N) \text{ conv } N$$

It follows that the identity function is λ-definable. The formula $S \longrightarrow \lambda pfx \cdot f(p(f,x))$ λ-defines the successor function. In fact,

$$S(N) \longrightarrow \{\lambda pfx \cdot f(p(f,x))\}(N)$$

$$\longrightarrow \{\lambda p[\lambda fx \cdot f(p(f,x))]\}(N)$$

$$\text{conv } S_N^p \; |\lambda fx \cdot f(p(f,x))| \text{ which is}$$

$$\lambda fx \cdot f(N(f,x)) \text{ conv}$$

$$\lambda fx \cdot f(\{\lambda ab \cdot \underbrace{a(\ldots a}_{n+1 \text{ times}}(b) \ldots)\}(f,x))$$

$$\text{conv } \lambda fx \cdot \underbrace{f(f \ldots}_{n+2 \text{ times}} f(x) \ldots) \text{ which}$$

is the successor of n.

One more example is sufficient to illustrate the method. Consider the formula $A \longrightarrow \lambda abfx \cdot a(f,b(f,x))$. That A λ-defines the function $\emptyset(n_1, n_2) = n_1 + n_2 + 1$ can be seen as follows.

$$A(N_1, N_2) \text{ conv } \lambda fx \cdot N_1(f, N_2(f,x))$$

$$\text{conv } \lambda fx \cdot \{\lambda ab \cdot \underbrace{a(\ldots a}_{n_1+1 \text{ times}}(b) \ldots)\}(f, N_2(f,x))$$

conv $\lambda fx \cdot \underbrace{f(\ldots f}_{n_1+1 \text{ times}}(N_2(f,x)) \ldots)$

conv $\lambda fx \cdot \underbrace{f(\ldots f}_{n_1+1 \text{ times}}(\{\lambda ab \cdot \underbrace{a(\ldots a}_{n_2+1 \text{ times}}(b) \ldots)\}(f,x)) \ldots)$

conv $\lambda fx \cdot \underbrace{f(\ldots f}_{n_1+1 \text{ times}}(\underbrace{f \ldots f}_{n_2+1 \text{ times}}(x) \ldots) \ldots)$

which is $fx \cdot \underbrace{f(\ldots f}_{n_1+n_2+2 \text{ times}}(x) \ldots)$

In Chapter I, the question of whether or not it is possible to find algorithmic solutions to certain classes of mathematical problems was partially answered by applying the result that certain sets of non-negative integers are not recursive to the problem of determining, for example, whether or not an equation of the form $x^n + y^n = z^n$ has a solution in integers. A somewhat analogous result (due to Church [1]) will now be derived in terms of this and the last chapter. First, by reasoning similar to that in Chapter I, the following assumption is adopted.

T': Every constructible (or general recursive) function is λ-definable.

In order that the results of Chapter I may be applied, a representation of w.f.f.s by non-negative integers is

defined. A number is associated with each symbol in a given formula as follows.

λ : 1

$\{$, (, [: 11

$\}$,),] : 13

The i-th variable in order of appearance in the formula: p_{i+6}, the $(i + 6)$-th prime number.

Definition: If A is a formula consisting of the sequence of symbols $\alpha_1, \ldots, \alpha_n$ and $t_1, \ldots, t_n$ are the corresponding numbers, then the number $2^{t_1} \cdot 3^{t_2} \cdot 5^{t_3} \ldots p_n^{t_n}$ (where p_n is the n-th prime) is the Gödel representation of A.

As an example, consider the formula

$$\lambda fx\ f(x) \longrightarrow \lambda f[\ \lambda x[\ \{f\}\ (x)]]$$

whose Gödel representation is

$$2^1 \cdot 3^{17} \cdot 5^{11} \cdot 7^1 \cdot 11^{19} \cdot 13^{11} \cdot 17^{11} \cdot 19^{17} \cdot 23^{13} \cdot 29^{11} \cdot 31^{19} \cdot 37^{13} \cdot 41^{13} \cdot 43^{13}.$$

The following definition restricts attention to those formulas which are relevant to the present purpose (this point will be discussed later).

Definition: A formula A has a normal form if there is a formula B such that A conv B and B has no part of the form $\{\lambda x[M]\}(N)$.

The next three lemmas are consequences of the application of general recursive function theory to Gödel representations. Their rather lengthy proofs were carried out by Kleene [4].

Lemma A: If a formula has a normal form, every well formed part of it has a normal form.

Lemma B: The set of positive integers which are Gödel representations of w.f.f.s which have a normal form is r.e.

Lemma C: Given a formula A, and a positive integer n, there exists a general recursive function of two variables $\phi(m,n)$ such that if m is the Gödel representation of A, $\phi(m,1)$, $\phi(m,2)$, ... is a recursive enumeration of the Gödel representations of all formulas B such that

A conv B.

The main result now follows.

Theorem: The set R of non-negative integers which are Gödel representations of w.f.f.s that have a normal form is not recursive.

Proof: Suppose the contrary and, by lemma B, let m_1, m_2, ... be a recursive enumeration of all Gödel representations of w.f.f.s A_1, A_2, ... which have a normal form. Define the function $\emptyset(n)$ by

$$\emptyset(n) = \begin{cases} 1 \text{ if } \{A_n\}(N) \text{ is not convertible into any of the formulas } 1, 2, 3, \ldots \\ n_1 + 1 \text{ if } \{A_n\}(N) \text{ conv } N_1 \ (N_1 \text{ one of } 1, 2, 3, \ldots). \end{cases}$$

Then $\emptyset$ is constructible (general recursive) and hence by T' λ-definable by a formula P. In fact, if B is a w.f.f. and m is its Gödel representation, then, by the assumption that R is recursive, the characteristic function of R is constructible (general recursive). If then, B has a normal form, lemma C provides that it can be constructively determined and compared with the formulas

1, 2, 3, ... none of which has a part of the form $\{\lambda x[M]\}(N)$. Thus, by lemma A, P has a normal form. But P is not any of A_1, A_2, ... since, for each n, P(N) is not convertible into A(N).

This theorem is significant because of the following result obtained by Kleen [2]. Suppose $\phi_i(n_1, \ldots, n_p)$ and $\psi_i(n_1, \ldots, n_p)$, $i = 1, \ldots, m$ are defined for all p-tuples of positive integers $n_1, \ldots, n_p$ and take values which are positive integers. If ϕ_i and ψ_i are λ-definable for each $i = 1, \ldots, m$, then there can be found a formula L such that

(1) if solutions of the system

$$\phi_i(n_1, \ldots, n_p) = \psi_i(n_1, \ldots, n_p) \quad i = 1, \ldots, m$$

exist, L(1), L(2), ... are convertible into B_1, B_2, ... respectively where each B_i is a formula which represents an n-tuple $n_{1_i}, \ldots, n_{p_i}$ which is a solution of the system, and every solution of the system is represented by some B_i;

(2) if less than n different solutions exist, L(N) does not have a normal form.

For example, a formula L can be found such that

(1) L generates the solutions of $x^t + y^t = z^t$ in positive integers if such solutions exist and

(2) the problem of whether $x^t + y^t = z^t$ has at least one solution in integers is equivalent to the problem of whether L(1) has a normal form.

CHAPTER III

COMPUTABILITY

Yet another possible method for giving precise meaning to the intuitive notion of constructibility was proposed by Turing [8]. In terms of the previous chapters, Turing's idea is to identify a constructible function as one whose value for any given argument can be computed by a machine. What is of course needed is an abstract definition of "computing machine". The "machines" which will be defined in this chapter are called Turing machines.

Definition: A Turing machine is a matrix of the form

	q_1	q_2	...	q_n
S_1	X_{11}	X_{12}	...	X_{1n}
S_2	X_{21}	X_{22}	...	X_{2n}
⋮	⋮	⋮	⋮⋮⋮	⋮
S_m	X_{m1}	X_{m2}	...	X_{mn}

where each X_{kp} is in one of the forms S_iLq_j, S_iSq_j S_iRq_j !, $(i = 1, \ldots, m ; j = 1, \ldots, n)$. It is also possible for some of the boxes in the matrix to be empty. The set $S_1, \ldots, S_m$ is the external alphabet of the machine. Each column q_i, $i = 1, \ldots, n$ is called a state or logical unit of the Turing machine.

The operation of a Turing machine may be described as follows. The memory unit is conceived of as a tape, infinitely long in both directions and divided into cells. Each cell can contain at most one symbol of the external alphabet. The initial information is given to the machine in the form of finite strings of symbols of the external alphabet on the tape. The machine operates in cycles, and at the end of each cycle, all of the information on the tape constitutes the intermediate information at that stage.

Definition: Given some initial information I expressed in the external alphabet of the machine, there are two possible cases:

(1) If after a finite number of cycles, the machine halts, the machine is said to be applicable to the initial information I, which has been transformed into R, the information appearing on the tape after the last

cycle.

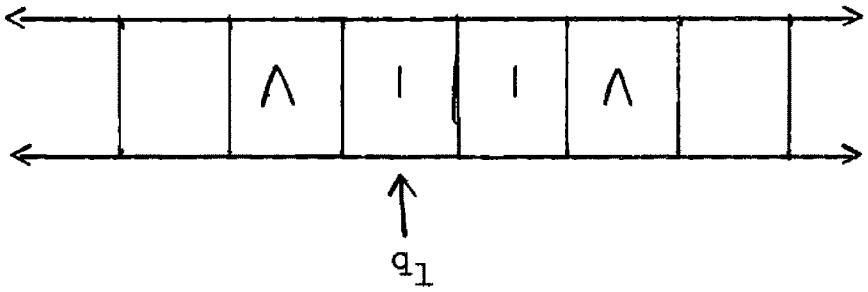
(2) if the machine never halts, it is inapplicable to the information I.

Definition: A turing machine can solve a given class of problems if it is applicable to the information representing (in some fixed code using the external alphabet of the machine) any problem of the class, and if it transforms this information into the information representing the solution (in the same code).

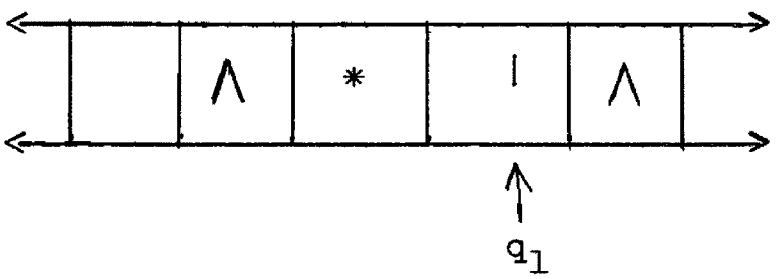
An example serves best to illustrate the meaning of "cycle" and "operation of the machine". Consider the Turing machine

	q_1	q_2
Λ	Sq_2	Lq_1
1	$*Rq_1$	
*	αSq_2	
α		!

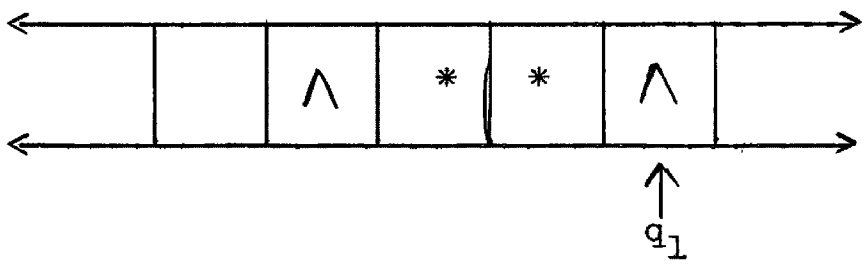
and let the initial information on the tape be



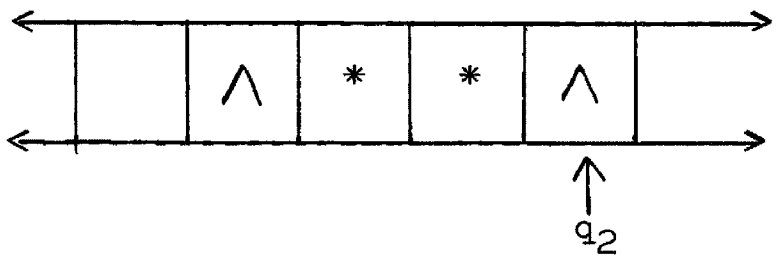
The initial information includes the starting position of the machine, the arrow indicating the cell at which the machine executes the first cycle. The machine begins in state q_1 and scans the symbol |. The entry in the matrix corresponding to the pair (|, q_1) is $*Rq_1$. This means that, in this cycle, the symbol | is changed to the symbol *, the machine prepares to scan the cell immediately to the right and enters into state q_1. Thus, at the end of the first cycle, we have



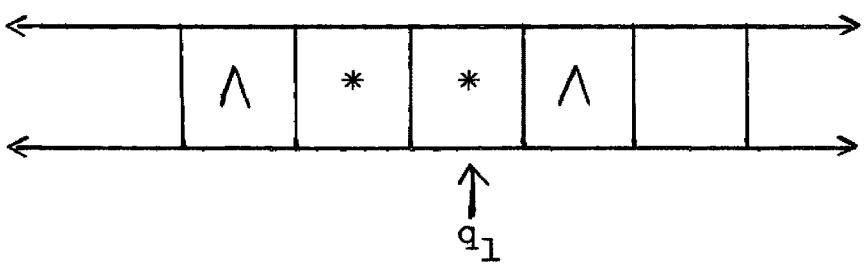
Likewise, at the end of the second cycle, we have



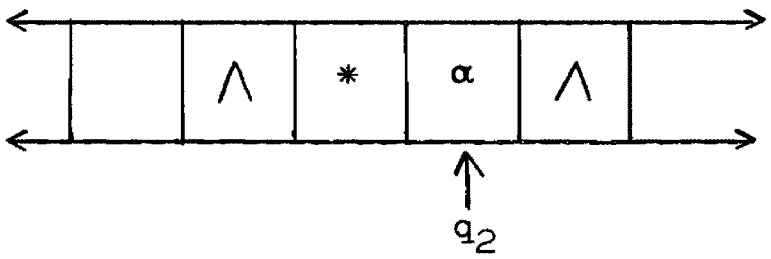
For the third cycle, the pair (Λ , q_1) corresponds to the entry Λ Sq_2 : the symbol Λ is left unchanged, the machine prepares to scan the same cell, and enters into state q_2. At the end of this cycle we have



Now (Λ , q_2) corresponds to Λ Lq_1 : the symbol is left unchanged, the machine prepares to scan the cell immediately to the left and enters state q_1 and we have

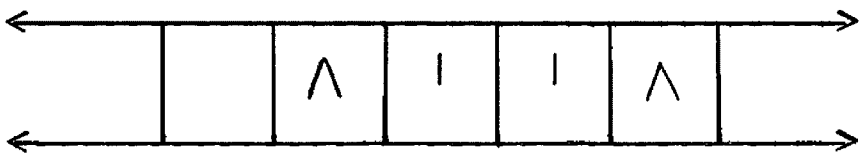


The next cycle gives rise to

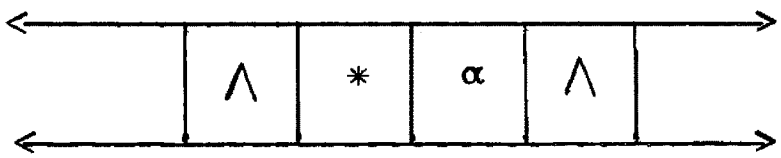


Finally, the pair (α, q_2) corresponds to !, which is used to indicate the halt order.

This Turing machine is, then, applicable to the information



and transforms it into



(Actually, this machine is applicable to any finite string of strokes bounded on each end by Λ).

Note that, if the entry corresponding to the pair $(|, q_1)$

is changed to $|Sq_1$, then the new Turing machine is not applicable to the initial information. Also note that what appears in the blank boxes is irrelevant with this initial information.

It will now be shown how Turing machines can be defined to carry out some of the simple arithmetical algorithms. In terms of the previous chapters, this may be interpreted as determining the value (the terminal information on the tape) of a function for a given argument (initial information).

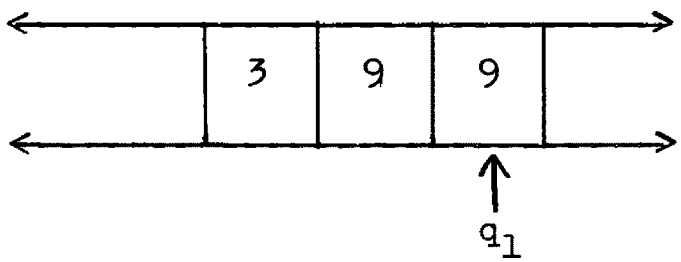
Given a positive integer n in decimal form, the following Turing machine transforms it into the successor of n, also in decimal form.

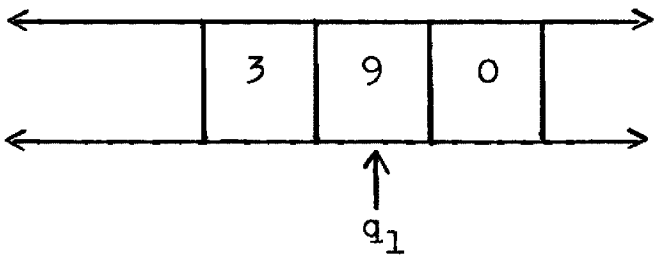
(see next page)

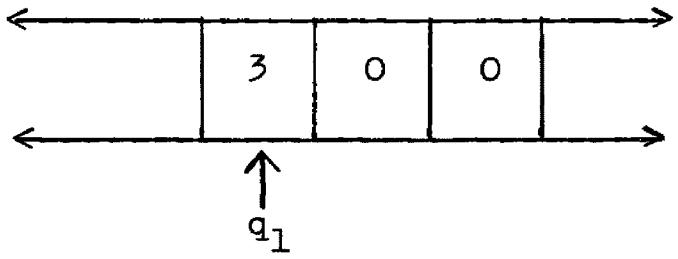
	q_1	q_2
0	$1Sq_2$	!
1	$2Sq_2$	!
2	$3Sq_2$	!
3	$4Sq_2$	!
4	$5Sq_2$	!
5	$6Sq_2$	!
6	$7Sq_2$	!
7	$8Sq_2$	!
8	$9Sq_2$	!
9	$0Lq_1$	!
Λ	$1Sq_2$	!

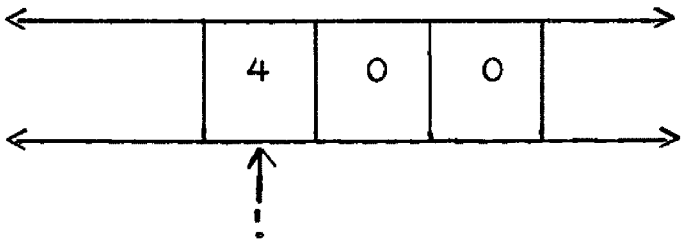
where Λ represents an empty cell.

The tape configurations for n = 399 are

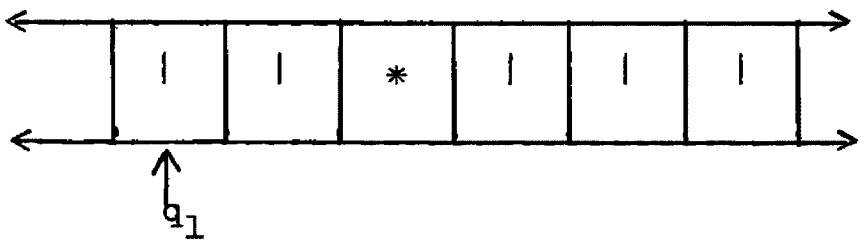




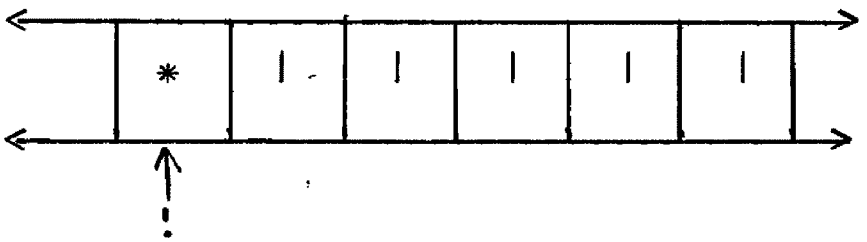




A Turing machine can be designed to add two positive integers if each integer is represented as a series of consecutive strokes. For example, for the computation $2 + 3 = 5$, the initial information is



and is transformed into



The Turing machine which performs such additions is

	q_1	q_2	q_3
|	$\wedge Rq_3$	$\|Lq_2$	$\|Rq_3$
$\wedge$	$\wedge Rq_1$	$\wedge Rq_1$	$\|Sq_2$
*	!	$*Lq_2$	$*Rq_3$

where $\wedge$ represents an empty cell.

A slightly more complicated Turing machine may be defined which converts two positive integers in decimal form into their sum, also in decimal form. The machine would transform each decimal form into the correct nu,-ber of consecutive strokes, add as above, and transform the new string into its decimal equivalent.

In this chapter, the assumption corresponding to T and T' of chapters I and II is

T": Every constructible function is computable. That is, given any constructible function f(n), a Turing machine can be defined which produces on the tape a sequence of 0's and 1's such that the number of 1's between the (n-1)-th 0 and the n-th 0 is f(n).

This statement, when written in the following form, is the basic hypothesis of the theory of algorithms.

"Every algorithm can be carried out by some Turing machine"

Following the same development as in Chapters I and II, it will now be shown that the concept of Turing machines as the formal counterpart of constructibility

also gives rise to constructively (or algorithmically) unsolvable problems.

Definition: Suppose that on the tape of a Turing machine there appears the machine's own defining matrix in some code written in the machine's own external alphabet. If the machine is applicable to this initial information, it is called self-applicable. Otherwise, it is non-self-applicable.

Definition: The self-computability problem is: Given any coded defining matrix of some Turing machine, determine whether the corresponding machine is self-applicable.

Assuming the basic hypothesis of the theory of algorithms, the proof of the following theorem will be sketched (Trakhtenbrot [7]).

Theorem: The self-computability problem is algorithmically unsolvable.

Proof: Assume that there exists a machine A which solves the self-computability problem. Then A can be thought of as capable of transforming every defining matrix of a

self-applicable machine into the symbol σ and every defining matrix of a non-self-applicable machine into the symbol β. A is now modified so that when the symbol σ appears on its tape, the machine repeatedly scans the symbol σ without altering it. The new Turing machine B is then applicable to all defining matrices of non-self-applicable machines and inapplicable to defining matrices of self-applicable machines. But then B is self-applicable if, and only if, B is non-self-applicable.

CHAPTER IV

THE EQUIVALENCE OF GENERAL RECURSIVENESS, DEFINABILITY AND COMPUTABILITY

The assumptions T, T', T'' of chapters I, II, and III identified the intuitive concept of constructible function with the formal definitions of general recursive, λ-definable, and computable respectively. These three formal definitions are then equivalent under the assumptions T, T', T'', and this equivalence is a necessary condition for the validity of T, T', and T''. In this chapter, the equivalence is discussed independent of T, T', T'' with the following theorems.

Theorem A: Every general recursive function is λ-definable.

Theorem B: Every λ-definable function of one variable which takes values of 0 or 1 is computable.

Theorem C: Every computable function of one variable whose range is the set of positive integers is general recursive.

The proof of theorem A (Kleene [4]) consists of showing that the initial functions I-1, I-2, I-3; the schemes S-1, S-2; and $\varepsilon\, n[P(n)]$ are λ-definable and hence that every general recursive function is λ-definable.

I'-1: If $n_1 = n + 1$, $n_1 \longrightarrow N_1$, $n \longrightarrow N$, then
$S(N) \longrightarrow \{\lambda pfx \cdot f(p(f,x))\}$ (N) conv N_1

I'-2: $\{\lambda t \cdot t(\lambda a[a], N)\}$ (N') conv N where N' is one of the formulas 0, 1, 2, ...

I'-3: $\{\lambda t_1 \ldots t_n \cdot t_1(I, \ldots, t_n(I, t_i) \ldots)\}$
$(N_1, \ldots, N_k)$ conv N_i where $I \longrightarrow \lambda a[a]$

S'-1: $\{\lambda n_1 \ldots n_k \cdot G(H_1(n_1, \ldots, n_k), \ldots,$
$H_m(n_1, \ldots n_k)\}$ $(N_1, \ldots, N_k)$ conv
$G(H_1(N_1, \ldots, N_k), \ldots H_m(N_1, \ldots N_k))$

This follows immediately from operation II on w.f.f.s.

S'-2: If G and H are formulas with no free variables (formulas which λ-define functions have no free variables) there is a formula L such that

$$L(0, N_2, \ldots, N_k) \text{ conv } G(N_2, \ldots, N_k)$$
$$L(S(N), N_2, \ldots, N_k) \text{ conv}$$
$$H(N, L(N, N_2, \ldots, N_k), N_2, \ldots, N_k)$$

The proof of the existence of such a formula L is somewhat complicated and lengthy in detail, but the method can be illustrated as follows. The following lemmas are assumed without proof.

Lemma 1: The functions $\min(n_1, n_2)$ and $P(n)$, the

predecessor of n, are λ-definable.

Lemma 2: Let $-1 \longrightarrow \lambda fx \cdot x(f)$. Then, if A_{-1}, A_0, A_1 have no free variables, there is a formula F such that $F(N)$ conv $A_n (n = 0, 1; n \longrightarrow N)$ and $f(-1)$ conv A_{-1}.

Lemma 3: If F has no free variables, there is a formula L such that $L(N)$ conv $F(N,L)$ and $L(-1)$ conv I.

Now, given formulas G and H with no free variables, choose K by lemma 2 so that

$$K(0) \text{ conv } \lambda yf \cdot y(f(-1), G) \text{ and } K(1) \text{ conv } \lambda yfx_2 \ldots x_k \cdot H(P(y), f(P(y), x_2, \ldots x_k), x_2, \ldots, x_k)$$

Let $F \longrightarrow \lambda y \cdot K(\min(y, 1), y)$. Then the required formula L is given by lemma 3.

Finally, if R λ-defines the function $p(n_1, \ldots, n_k, n)$ and for each k-tuple $n_1, \ldots, n_k$ there is at least non-negative integer n such that $p(n_1, \ldots, n_k, n) = 0$ then there is a formula E such that $E(R)$ λ-defines $\varepsilon n[p(n_1, \ldots, n_k, n) = 0]$. To establish this, choose K by lemma 2 so that

$K(0)$ conv $\lambda fyr \cdot r(y, f(-1), y)$ and

$K(1)$ conv $\lambda fyr \cdot f(r(S(y)), S(y), r)$

Let $F \longrightarrow \lambda x \cdot K(\min(x, 1))$ and choose L by lemma 3.

Let $E \longrightarrow \lambda r x_1 \ldots x_k \cdot L(r(x_1, \ldots, x_k, 0), 0, r(x_1, \ldots, x_k))$.

The proof of theorem B (Turing [9]) consists of constructing a Turing machine which, given a formula F (that λ-defines a function f) as the initial information, transforms it into a sequence of 0's and 1's such that the n-th term of the sequence is 0 if F(N) conv 0 and 1 if F(N) conv 1. The detailed description of the machine appears in Turing's paper referred to above. The general idea is to construct a machine M which obtains successively every formula into which a given formula is convertible. This machine is enlarged into another machine M which contains M_1 as a part. The machine M operates in stages. At the n-th stage, the formula F(N) is formed and is supplied to M_1 which converts it successively into other formulas. Each formula into which F(N) is convertible eventually appears and, as it does, it is compared with the formulas 0 and 1. If it is identical with the first, the machine prints a 0, if with the second, a 1 and, if neither, M_1 continues. Since F(N) conv 0 or F(N) conv 1 (it can convert into just one formula

representing a numeral since F λ-defines f), M will eventually print 0 or 1, the n-th stage will be completed, and the (n + 1)-th stage begun.

Theorem B can be generalized to include all λ-definable functions whose range is the set of positive integers by showing that for any λ-definable function which takes positive integer values there exists a λ-definable function $\phi(n)$ with the property that $\phi(n)$ is either 0 or 1 and the number of 1's between the (k-1)-th and the k-th 0 is $\phi(k)$. Consider the function

$$\phi(n) = Sg(\min_{k=1,\ldots,n} |n - (\sum_{i=1}^{k} f(i) + k)|)$$

where

$$Sg(x) = \begin{cases} 1 & \text{if } x \neq 0 \\ 0 & \text{if } x = 0 \end{cases}$$

The function ϕ is general recursive (Kleene [5]) and hence by theorem A, ϕ is λ-definable. That ϕ has the desired properties can be seen by noting that $\phi(m) = 0$ if m is of the form $k-1 + \sum_{i=1}^{k=1} f(i)$ and $\phi(m) = 1$ otherwise. If, then, m is of the form $k-1 + \sum_{i=1}^{k-1} f(i)$, k-1 is the number of integers $j \leq m$ such that $\phi(j) = 0$ and $\sum_{i=1}^{k=1} f(i)$ is the number of integers $p \leq m$ such that $\phi(p) = 1$. It is exactly under these conditions that $\phi(m)$ must be 0.

The general idea of the proof of theorem C (Turing

[9]) is as follows. Suppose that f(n) is a computable function. The complete description or configuration of the Turing machine which computes f is uniquely described, at the end of a given cycle, by (1) the symbols appearing on the tape, (2) the symbol being scanned, and (3) the state the machine is in at the end of that cycle. Furthermore, the complete configuration U_n at the end of the n-th cycle uniquely determines the complete configuration U_{n+1} at the end of the (n+1)-th cycle. By using a technique very similar to Gödel numbering, each complete configuration U_n can be described by a single positive integer m_n which is computed in terms of the three factors mentioned above. It can then be shown that the function which gives m_{n+1} in terms of m_n is general recursive. This is used to define a general recursive function $\theta(n)$ such that

$\theta(n) = 0$ if in going from the (n+1)-th to the (n+2)-th complete configuration the machine prints a 0,

$\theta(n) = 1$ if it prints a 1

$\theta(n) = 2$ otherwise

Define the general recursive functions σ, $\mathbb{U}$, $\emptyset$ by

$$\begin{cases} \sigma(0) = 0 \\ \sigma(n') = [Sg(\theta(n))]\cdot\sigma(n) + [\overline{Sg}(\theta(n))]\cdot(\sigma(n))' \end{cases}$$

where

$$\overline{Sg}(x) = \begin{cases} 0 \text{ if } x \neq 0 \\ \\ 1 \text{ if } x = 0 \end{cases} ;$$

$$\Psi(n,m) = \max(0,\ n - \sigma(m));$$

$$\begin{cases} \phi(0) = 0 \\ \phi(n') = Sg(\theta(n))\cdot[Sg(|\theta(n)-1|)\cdot\phi(n) \\ \qquad\qquad + \overline{Sg}(|\theta(n)-1|)\cdot(\phi(n))'] \end{cases}$$

Then $f(n) = \phi(\varepsilon\, m[\Psi(n,m) = 0])$ and hence $f(n)$ is general recursive.

Theorems A, B, and C, together with the generalization of B can be summarized as follows:

Given a function $f(n)$ with domain the set of non-negative integers and range the set of positive integers, the following are mutually equivalent.

1) $f(n)$ is general recursive.

2) $f(n)$ is λ-definable.

3) $f(n)$ is computable.

REFERENCES

1. Alonzo Church, "An unsolvable problem of elementary number theory", the American Journal of Mathematics, Vol. 58, pp. 345-363 (1936).
2. S. C. Kleene, "A theory of positive integers in formal logic", the American Journal of Mathematics, Vol. 57, pp 153-173, 219-244 (1939).
3. S. C. Kleene, "General recursive functions of natural numbers", Mathematische Annalen, Band 112, Heft 5 pp. 727-742 (1936).
4. S. C. Kleene, " λ -definability and recursiveness", the Duke Mathematical Journal, Vol. 2, pp 340-393 (1936).
5. S. C. Kleene, Introduction to Metamathematics, D. Van Nostrand Company, Inc., New York 1952.
6. Emil Post, "Recursively enumerable sets of positive integers and their decision problems", the Bulletin of the American Mathematical Society, Vol. 50, pp. 284-316 (1944).
7. B. A. Trakhtenbrot, Algorithms and Automatic Computing Machines, D. C. Heath and Company, Boston, 1963.
8. A. M. Turing, "On computable numbers, with a application to the entscheidungsproblem", Proceedings of

the London Mathematical Society, ser. 2, Vol. 42, pp. 230-265 (1937).

9. A. M. Turing, "Computability and λ-definability", the Journal of Symbolic Logic, Vol. 2, No. 4, pp. 153-163 (1937).